

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**Method and Apparatus for Correlating Events**

Inventors:

Lev Novik

Patrick R. Kenny

Alexander E. Nosov

ATTORNEY'S DOCKET NO. MS1-694US

094754-0301  
P.050-4294860

1 **TECHNICAL FIELD**

2 The present invention relates to computing systems and, more particularly,  
3 to the correlation of various events generated throughout a computing  
4 environment.

5  
6 **BACKGROUND**

7 Computer systems, such as servers and desktop personal computers, are  
8 expected to operate without constant monitoring. These computer systems  
9 typically perform various tasks without the user's knowledge. When performing  
10 these tasks, the computer system often encounters events that require a particular  
11 action (such as logging the event, generating an alert for a particular system or  
12 application, or performing an action in response to the event). Various  
13 mechanisms are available to handle these events.

14 A computing enterprise typically includes one or more networks, services,  
15 and systems that exchange data and other information with one another. The  
16 enterprise may include one or more security mechanisms to safeguard data and  
17 authenticate users and may utilize one or more different data transmission  
18 protocols. At any particular time, one or more networks, services or systems may  
19 be down (e.g., powered down or disconnected from one or more networks).  
20 Networks, services or systems can be down for scheduled maintenance, upgrades,  
21 overload or failure. Application programs attempting to obtain event data must  
22 contend with the various networks, services, and systems in the enterprise when  
23 they are down. Additionally, application programs must contend with the security  
24 and network topology limitations of the enterprise as well as the various protocols  
25

used in the enterprise.

Operating system components, services, and applications generate a variety of different events. A particular component or application may request to be informed of a particular event (e.g., when a server crashes or when a user logs on to the system). Other components or applications may want to be notified when a particular series of events occur within a particular time period. For example, a network administrator may want to know when a server crashes within three seconds of a user logging into the system. Server crashes alone may be relatively common and user logins may also be common such that the network administrator is not particularly interested in either event by itself. However, when these two events occur within a few seconds of one another, there may be a relationship between the two events (e.g., the user login was at least partially responsible for the server crash).

Existing systems provide predefined functions that allow a network administrator or other user to create a relationship between two events. This relationship between two events is commonly referred to as a "correlation" between the two events. The predefined correlation functions provided by existing systems require the user to select from one of the predefined functions. If the correlation function desired by the user has not already been created, the user must request that the developer or supplier of the functions create a new function to meet the user's needs. If the developer or supplier is willing to create a new correlation function, this custom development work may be very expensive. Depending on the expected demand for the new correlation function, the developer or supplier may not be willing to create the requested function.

1 If the developer is unwilling to create a new correlation function or the cost  
2 is too high, the user can attempt to use an existing correlation function that is  
3 "closest" to the user's requirements. Such a solution may result in a significant  
4 number of unwanted event notifications or may result in a failure to notify the user  
5 of a desired sequence of events.

6 The system and method described herein addresses these limitations by  
7 providing a flexible correlation system and method that allows a user to correlate  
8 multiple events and/or data.

## 9 10 11 **SUMMARY**

12 The correlation system and method described herein supports the  
13 correlation of multiple events as well as the correlation of one or more events with  
14 one or more data elements. A flexible programming model is used to correlate  
15 events and/or data. This programming model allows for the creation of many  
16 different commonly used correlation functions as well as the creation of  
17 specialized correlation functions to meet the specific needs of a user. The  
18 predefined correlation functions allow the user to enter the particular event and/or  
19 data parameters to be monitored. These predefined correlation functions can be  
20 used without understanding the underlying programming model. However, a user  
21 with an understanding of the programming model is able to create custom  
22 correlation functions to meet particular needs. Thus, the user is not limited to  
23 using a particular set of predefined correlation functions.

24 The programming model provides a way to perform correlation by allowing  
25 the user to define state classes (e.g., a schema), configure actions that change the

1 state (e.g., using an updating consumer and a programming language), and a way  
2 to link the occurrence of events to these actions.

3 In one embodiment, multiple events are received and applied to a  
4 correlation function. The correlation function is implemented as a state machine.  
5 A specific event is generated if the correlation function is satisfied by the multiple  
6 received events.

7 A described embodiment continues to receive additional events and apply  
8 the additional events to the correlation function if the correlation function is not  
9 satisfied by the multiple received events.

10 In another embodiment, multiple data elements are also received and  
11 applied to the correlation function.

12 A particular embodiment identifies a schema for creating state machines  
13 that correlate at least two events. An instance of a particular state machine is  
14 created and transitions for the particular state machine are defined by subscribing  
15 to at least one event. An update consumer is applied to the particular state  
16 machine to update the state of the particular state machine.

17 In a described embodiment, the particular state machine is deleted if the  
18 particular state machine reaches a final state.

19 In another embodiment, the particular state machine includes a timer such  
20 that the particular state machine is deleted if the timer expires.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

Fig. 1 illustrates a block diagram of a system that receives event information from multiple event providers and provides event information to multiple event consumers.

Fig. 2 is a flow diagram illustrating an event-handling procedure.

Fig. 3 illustrates a block diagram of a system that receives multiple events from various event sources, filters the events, and distributes the filtered events to multiple event consumers.

Fig. 4 is a flow diagram illustrating a procedure for handling events in the system of Fig. 3.

Fig. 5 illustrates a block diagram of an exemplary system in which a correlator receives events, data, and correlation functions from multiple sources.

Fig. 6 is a flow diagram illustrating a procedure for correlating events and/or data in an enterprise.

Fig. 7 is a flow diagram illustrating a procedure for implementing a state machine that applies a correlation function.

Fig. 8 illustrates an example of a suitable operating environment in which the event distribution and event handling system and method may be implemented.

## DETAILED DESCRIPTION

The system and method described herein provides for the correlation of multiple events and/or data using a predefined set of correlation functions developed using a schema, which defines state classes and permits the use of a variety of different programming languages. The selected programming language is used to create state machines that implement correlation functions. Predefined state machines can be utilized without understanding the underlying programming language used to create the state machines. With an understanding of the programming language, custom state machines can be created to implement correlation functions that are not supported by the predefined set of state machines. The state machines can be applied to events generated by any event source in an enterprise. Similarly, state machines may correlate events from any event source with data from any data source in the enterprise. This programming model (i.e., the combination of the schema with the programming language) provides a way to perform correlation by allowing the user to define state classes, configure actions that change the state, and link the occurrence of events to these actions.

Web-Based Enterprise Management (WBEM) provides uniform access to management information throughout an enterprise. WBEM is an industry initiative to develop technology for accessing management information in an enterprise environment. This management information includes, for example, information on the state of system memory, inventories of currently installed client applications, and other information related to the status of the system. A particular embodiment of the event-handling system is represented using Windows

1 Management Instrumentation (WMI) developed by Microsoft Corporation of  
2 Redmond, Washington, which provides an infrastructure to handle various events  
3 generated by event sources throughout an enterprise.

4 WMI technology enables systems, applications, networks, and other  
5 managed components to be represented using the Common Information Model  
6 (CIM) designed by the Distributed Management Task Force (DMTF). This model  
7 is used to perform correlation functions discussed herein. CIM is an extensible  
8 data model for representing objects that exist in typical management  
9 environments. CIM is able to model anything in the managed environment,  
10 regardless of the location of the data source. The Managed Object Format (MOF)  
11 language is used to define and store modeled data. In addition to data modeling,  
12 WMI provides a set of base services that include query-based information retrieval  
13 and event notification. Access to these services and to the management data is  
14 provided through a single programming interface.

15 WMI classes define the basic units of management. Each WMI class is a  
16 template for a type of managed object. For example, Win32\_DiskDrive is a model  
17 representing a physical disk drive. For each physical disk drive that exists, there is  
18 an instance of the Win32\_DiskDrive class. WMI classes may contain properties,  
19 which describe the data of the class and methods, which describe the behavior of  
20 the class.

21 WMI classes describe managed objects that are independent of a particular  
22 implementation or technology. WMI includes an eventing subsystem that follows  
23 the publish-subscribe model, in which an event consumer subscribes for a  
24 selection of events (generated by one or more event providers) and performs an  
25



1 action as a result of receiving the event. WMI also provides a centralized  
2 mechanism for collecting and storing event data. This stored event data is  
3 accessible by other systems via WMI tools and/or application programming  
4 interfaces (APIs).

5 Although particular embodiments are discussed herein as using WMI,  
6 alternate embodiments may utilize any enterprise management system or  
7 application, whether web-based or otherwise. The event providers and event  
8 consumers discussed herein are selected for purposes of explanation. The  
9 teachings of the present invention can be used with any type of event provider and  
10 any type of event consumer. Additionally, the event-handling system and method  
11 described herein can be applied to any type of enterprise or other arrangement of  
12 computing devices, applications, and/or networks.

13 Fig. 1 illustrates a block diagram of a system 100 that receives event  
14 information from multiple event providers 114 (i.e., event sources) and provides  
15 event information to multiple event consumers 102 (i.e., the users of the event  
16 data). System 100 includes a WMI module 106, which receives event data from  
17 multiple event sources 114 and receives requests for information (e.g., notification  
18 of particular events) from multiple event consumers 102. The multiple event  
19 sources are identified as event providers 112. The multiple event consumers are  
20 identified as applications 104.

21 Event providers 112 include, for example, systems, services or applications  
22 that generate event data. An exemplary event provider is a disk drive (or an  
23 application that monitors the status of a disk drive). The disk drive may generate  
24 an event indicating the available storage capacity on the disk drive or indicating  
25

the amount of data currently stored on the disk drive. The disk drive may also generate an event indicating that the disk drive is nearly full of data (e.g., when ninety-five percent or more of the disk drive's capacity is used).

Event consumers 102 may request to be notified of certain events (also referred to as "subscribing" to an event). An example event consumer is an application that manages multiple storage devices in an enterprise. The application may request to receive events generated by any of the disk drives or other storage devices in the enterprise. The application can use this event information to distribute storage tasks among the multiple storage devices based on the available capacity of each device and/or the quantity of read or write requests received by each storage device.

System 100 also includes a set of policies 110, which are accessible by WMI module 106. Policies 110 may control the configuration of one or more systems in the enterprise. Other policies may define various activities, such as event filtering, event correlation, and the forwarding of events to particular devices or applications. A database 108 is coupled to WMI module 106. Database 108 stores various information related to the enterprise. For example, database 108 can store event data (i.e., creating an event log), policy data, and enterprise configuration information.

The WMI module 106 uses WMI features to provide a distributed architecture that is capable of selecting, filtering, correlating, forwarding, storing, and delivering event data in an enterprise. The WMI module also allows event consumers to request data related to a particular event, request data from a particular node or device in the enterprise, define the manner in which events are

1 correlated with one another, define how certain events should be forwarded, and  
2 define how to store event data.

3 The WMI module 106 provides a policy-based administration of the  
4 enterprise. The policy infrastructure allows administrators to set a policy in the  
5 Directory Service (DS) and the WMI module ensures that the proper set of WMI  
6 objects (e.g., filters, bindings, correlators, consumers, and configuration objects)  
7 are delivered to the proper devices or applications in the enterprise.

8 As shown in Fig 1, policies 110 and database 108 are separate from WMI  
9 module 106. However, in alternate embodiments, policies 110 and/or database  
10 108 may be integrated into WMI module 106.

11 Table 1 below identifies various types of event providers available in a  
12 particular embodiment. Additionally, the table includes a description of the events  
13 generated by each event provider. For example, the Win32 Provider generates  
14 events that include information related to the operating system, computer system,  
15 peripheral devices, file systems, and security for a particular device (such as a  
16 computer system) in the enterprise.

TABLE 1

Event Provider	Description of Events Provided
Win32 Provider	Supplies information about the operating system, computer system, peripheral devices, file systems, and security.
WDM Provider	Supplies low-level Windows Driver Model (WDM) information for user input devices, storage devices, network interfaces, and communications ports.
Performance Counter Provider	Exposes the raw performance counter information used to compute various performance values.
Windows Installer Provider	Supplies information about applications installed with the Windows Installer.

Fig. 2 is a flow diagram illustrating an event-handling procedure 200. The WMI module monitors event activity throughout the enterprise (block 202). The procedure 200 determines whether event data has been received from an event provider (block 204). If event data has been received, the WMI module records the event data and initiates any appropriate actions (block 206). An example action includes notifying an event consumer of the event (e.g., if the event consumer previously subscribed to such an event).

At block 208, the procedure 200 determines whether a new subscription for event information has been received. The procedure 200 may also determine whether a request to revise an existing subscription has been received. If a new subscription (or a revised subscription) is received, the procedure continues to block 210 where the WMI module retrieves the requested event information and provides the information to the requesting event customer. Alternatively, the procedure may log the subscription request and notify the requesting event

1 consumer when the next event is received that qualifies under the consumer's  
2 subscription request.

3 Fig. 3 illustrates a block diagram of a system 300 that receives multiple  
4 events from various event sources, filters the events, and distributes the filtered  
5 events to multiple event consumers. Multiple events 302 are received by an event  
6 filter 304 that determines which of the received events are passed through the filter  
7 to a correlator 310. The filter 304 applies various filter criteria 308 to the received  
8 events in determining which events pass through the filter to correlator 310.  
9 Additional details regarding correlator 310 and the correlation functions are  
10 discussed below.

11 The correlator 310 correlates various events and creates additional events  
12 312 that are provided to multiple filters 314, 320, 326, and 332. Each filter 314,  
13 320, 326, and 332 includes various filter criteria that determines what event  
14 characteristics are required to allow the event to pass through the filter. Although  
15 each event 312 is sent to all four filters, the event may be rejected (i.e., not pass  
16 through the filter) by any or all of the filters. Similarly, a particular event may  
17 pass through two or more different filters, depending on the filter criteria  
18 associated with each filter.

19 Each filter 314, 320, 326, and 332 is associated with a consumer (i.e., an  
20 event consumer) 316, 322, 328, and 334, respectively. For example, events that  
21 pass through filter 314 are provided to event logging consumer 316, which logs  
22 the event data to a storage device 318. The logged data can be retrieved at a later  
23 time for analysis or other purposes. Events that meet the criteria of filter 320 are  
24 provided to event forwarding consumer 322, which generates a forwarded event  
25 324 that is distributed to one or more destinations. Events that satisfy the criteria

1 of filter 326 are provided to mail consumer 328, which generates and sends an  
2 email message 330 in response to receipt of each event. The email message 330  
3 may contain information about one or more events (such as the event type or the  
4 source of the event). Events that pass through filter 332 are provided to scripting  
5 consumer 334, which executes a script that may perform a function and/or  
6 generate a script output 336.

7 Although the example of Fig. 3 illustrates four filters 314, 320, 326, and  
8 332 (and associated consumers 316, 322, 328, and 334, respectively) that receive  
9 events 312, alternate embodiments may include any number of filters and  
10 associated consumers. Further, the actions performed by consumers 316, 322,  
11 328, and 334 are provided as examples. Alternate consumers may perform any  
12 type of action in response to receiving an event.

13 Fig. 4 is a flow diagram illustrating a procedure 400 for handling events in  
14 the system of Fig. 3. An event is received by an event filter (block 402), such as  
15 filter 304 in Fig. 3. The procedure 400 determines whether the received event  
16 satisfies the event filter (block 404). Satisfying the event filter includes satisfying  
17 the filter criteria (e.g., filter criteria 308). If the received event does not satisfy the  
18 event filter, then the received event is discarded (block 406). Discarding an event  
19 may include ignoring the event or deleting the event and any reference to the event  
20 from storage registers or other storage mechanisms. If the received event satisfies  
21 the event filter (i.e., passes through the filter), a correlator correlates multiple  
22 received events (block 408) and may generate one or more new events that are  
23 provided to multiple event filters (e.g., filters 314, 320, 326, and 332 in Fig. 3) in  
24 block 410.  
25

Each event filter analyzes the event using its own filter criteria (block 412). Next, each event filter determines whether the event meets the event filter's criteria (block 414). This determination is performed by each event filter based on the filter criteria for that particular event filter. If the event does not meet the criteria for a particular event filter, that event filter discards the event (block 416). However, if the event satisfies the criteria for a particular event filter, that event filter passes the event to the event consumer that corresponds to the particular event filter (block 418). The event consumer then performs one or more actions based on the event (block 420). For example, the actions may include generating an email message or forwarding the event to another system. The procedure of Fig. 4 is repeated for each received event.

Fig. 5 illustrates a block diagram of an exemplary system 500 in which a correlator 502 receives events 504, data elements 506, and correlation functions 508 from multiple sources. Events 504 have passed through a filter (e.g., filter 304 in Fig. 3) prior to being received by correlator 502. The terms "data" and "data elements" are used interchangeably herein. Correlator 502 may be similar to the correlator 310 (Fig. 3) discussed above. Correlator 502 may receive events 504 directly from an event source or may receive events through an event filter, such as filter 304 discussed above. In one embodiment, correlator 502 receives specific event information based on event subscriptions applied to various event sources in the enterprise. Correlator 502 may receive data elements 506 directly from a data source or may receive the data through an intermediate device, such as a database or other data storage or data collection device. Correlator 502 also receives correlation functions from one or more sources, such as from an administrator's node. The correlation function 508 is a group of updating

1 consumers (discussed below) that receive events like other consumers. These  
2 updating consumers cause one or more events 510 to occur (either directly or  
3 indirectly).

4 Correlator 502 applies the received correlation functions to the events and  
5 data received from various sources throughout the enterprise. When the  
6 conditions of a particular correlation function are satisfied, correlator 502  
7 generates an event 510, which is distributed to various event consumers in the  
8 enterprise. In one embodiment, the event 510 is provided to event consumers that  
9 subscribed to receive that particular event.

10 Fig. 6 is a flow diagram illustrating a procedure 600 for correlating events  
11 and/or data in an enterprise. A correlator (e.g., correlator 502 in Fig. 5) receives  
12 events, data and correlation functions from multiple sources in an enterprise (block  
13 602). The received events and data are compared to the correlation functions  
14 (block 604). The procedure 600 next determines whether any of the correlation  
15 functions are satisfied by the received events and data (block 606). If not, the  
16 procedure 600 continues receiving events, data and correlation functions (block  
17 608), and returns to block 604 to continue comparing received events and data to  
18 the correlation functions. If any of the correlation functions are satisfied by the  
19 received events and data, the correlator generates one or more events based on the  
20 satisfied correlation function (block 610). After generating one or more events in  
21 block 610, the procedure 600 continues to block 608 to continue receiving events,  
22 data and correlation functions.

23 Fig. 7 is a flow diagram illustrating a procedure 700 for implementing a  
24 state machine that is capable of implementing a correlation function. Initially, a  
25 user defines or selects a pre-defined schema for the desired state machine (block



702). Next, an instance of the desired state machine is created (block 704). Transitions for the state machine are defined by subscribing to one or more events (block 706). An update consumer is applied to the state machine to update the state of the state machine (block 708). The updating consumer is applied to the state machine each time an event to which the updating consumer has subscribed occurs.

Procedure 700 then determines whether the state machine is in its final state (block 710). If the state machine is not in its final state, the procedure 700 returns to block 708 to again apply the update consumer to the state machine. If the state machine is in its final state, the procedure 700 continues to block 712, which deletes the current instance of the state machine. If another correlation function is to be implemented, a new instance of the desired state machine is created and executed.

Examples of events include a server crash, a user logging into the system, or a particular device becoming unavailable. Example data elements include the available disk space, the current memory utilization, and the number of users logged into particular servers. An example correlation function that correlates two events generates an email message when two different server crashes occur within five second of one another. An example correlation function that correlates an event with data generates an event when a server crashes and the available storage space on the server's hard drive is less than five megabytes. Another example correlation function pages an administrator when the available storage space on a server's hard disk stays below ten megabytes for at least five minutes. Any other selection of events and/or data can be combined to create a correlation function based on the desires of an administrator or other user.

As mentioned above, a state machine implements a desired correlation function that correlates events and/or data. A set of commonly used state machines are provided for use by administrators (or other users) in defining correlation functions. These commonly used state machines require the administrator to fill in certain parameters, but the administrator is not required to understand the programming language used to create the state machine. If the set of commonly used state machines does not include a state machine that performs the desired correlation function, a new state machine can be created using the appropriate programming language. The programming language can be any database language or other non-procedural language. In a particular embodiment, the programming language is SQL.

Each state machine is a class object. One or more instances of a state machine can be implemented simultaneously to monitor different events and data. In a particular enterprise, any number of instances of state machines may be operating simultaneously. In one embodiment of the invention, SQL is used to query various states in any state machine.

In a particular example, the schema for a state machine that detects a specific number of process crashes within a specified time period can be defined as follows.

```
Class StateA
{
    string ProcessName;
    int NumCrashes;
    int RemainingTime;
}
```

In the above example, the state machine is a class having three properties. ProcessName is a string that represents the name of the process being monitored for crashes. NumCrashes is an integer that identifies the number of crashes that will trigger an event if those crashes occur within a particular time period, which is defined by the property RemainingTime, which is an integer value. The RemainingTime property is reset each time a new instance of StateA is created. If RemainingTime reaches zero without being reset, an event is triggered indicating that the state machine time expired before detecting the specified number of crashes. When RemainingTime reaches zero, that particular instance of StateA is deleted because the specified parameters were not satisfied within the particular time period.

An administrator wanting to use the correlation function defined by StateA first creates an instance of StateA. The administrator then provides a value for NumCrashes and RemainingTime. Thus, the administrator need not understand the complete syntax of the state machine and need not understand the programming language used to define and create the state machine.

After defining the schema for the StateA state machine, the transitions for the state machine (i.e., the transitions from one state to another) are defined by subscribing to various events. Specifically, the transition is defined by the updating consumer and the event that causes the transition is defined by the event subscription. These event subscriptions function as the transitions for the state machine. When an appropriate event occurs, the state machine transitions to the next state. The state machine transitions are defined by identifying the event that will cause the transition and identifying the action to perform based on the occurrence of the event. The action may include, for example, generating an email

1 message, logging event data to a file, or forwarding an event to one or more  
2 destinations. The transitions are defined using updating consumer instances.

3 After defining the transitions for the state machine, an updating consumer is  
4 used to update the state of the state machine. The updating consumer (named  
5 “update”) is a class object. One or more instances of the updating consumer can  
6 be implemented simultaneously to handle the updating of different state machines.  
7 An example updating consumer implementation is illustrated below.

8  
9 Update StateA where ProcessName = ThisEvent.ProcessName  
10 set NumCrashes = NumCrashes + 1

11 In this example, the updating consumer updates an instance of state machine  
12 StateA, defined above. The ProcessName property is defined as  
13 “ThisEvent.ProcessName”, which inserts the name of the process that crashed  
14 (which is identified in the received crash event) as “ThisEvent”. The property  
15 NumCrashes is incremented by one each time a crash event is received.

16 While a particular state machine is operating, the various internal states of  
17 the state machine can be obtained (e.g., queried). This allows an administrator or  
18 other user to observe the correlation as the various events occur in a system. Even  
19 if the conditions have not yet been met to generate the appropriate event, the  
20 administrator can observe the current state or value of different properties (e.g.,  
21 how many crashes have occurred or how much time is left before the state  
22 machine is reset). The ability to observe the various states and properties of the  
23 state machine assists with troubleshooting and determining whether the desired  
24 correlation function has been properly established.  
25

Various examples have been discussed herein in which two different events are correlated with one another or an event is correlated with data. However, in alternate embodiments, any number of events can be combined together to form a correlation function. Similarly, any number of events can be combined with one or more data elements to create a correlation function.

The following example illustrates classes and class instances, a correlation scenario, updating consumers, filters and bindings as used with the present invention. Example class and instances of the class:

```
class ExampleClass
{
    [key] string Name;
    boolean Prop;
};

instance of ExampleClass
{
    Name = "A";
    Prop = TRUE;
};

instance of ExampleClass
{
    Name = "B";
    Prop = FALSE;
};
```

The correlation scenario is defined:

```
[ dynamic, provider("Microsoft WMI Transient Provider")]
class ExampleCorrelationState : MSFT_CorrelationStateBase
{
    boolean ReceivedEventA;
    boolean ReceivedEventB;
    [trns_egg_timer] uint32 Timer;
};
```

```

1      Class BothAandBEvent : MSFT_UCEventBase
2      {
3          string Name;
4      };
5
6      instance of MSFT_UpdatingConsumer as $UI
7      {
8          Id = "Initializer";
9          Scenario = "ExampleCorrelationScenario";
10         Commands = { "INSERT INTO ExampleCorrelationState "
11                     "( Id, Scenario, ReceivedEventA, ReceivedEventB, Timer ) "
12                     "( 'ExampleCorrelationState', 'ExampleCorrelationScenario', "
13                     " FALSE, FALSE, 0 )"};
14     };
15
16     instance of MSFT_UpdatingConsumer as $UA
17     {
18         Id = "SetEventA";
19         Scenario = "ExampleCorrelationScenario";
20         Commands = { "UPDATE ExampleCorrelationState "
21                     "SET ReceivedEventA = TRUE, Timer = 5 "
22                     "WHERE Scenario = 'ExampleCorrelationScenario' "};
23     };
24
25     instance of MSFT_UpdatingConsumer as $UB
26     {
27         Id = "SetEventB";
28         Scenario = "ExampleCorrelationScenario";
29         Commands = { "UPDATE ExampleCorrelationState "
30                     "SET ReceivedEventB = TRUE, Timer = 5 "
31                     "WHERE Scenario = 'ExampleCorrelationScenario' "};
32     };

```

The \$UA and \$UB updating consumers cause the timer to be reset to five seconds whenever either EventA or Event B occurs. The next updating consumer causes the ReceivedEventA and the ReceivedEventB to be reset when the timer expires.

```

1 instance of MSFT_UpdatingConsumer as $UTE
2 {
3     Id = "ResetTimer";
4     Scenario = "ExampleCorrelationScenario";
5     Commands = { "UPDATE ExampleCorrelationState "
6                   "SET ReceivedEventA = FALSE, ReceivedEventB = FALSE "
7                   "WHERE Scenario = 'ExampleCorrelationScenario' "};
8 };

```

The following defines filters and bindings to fully define the scenario:

```

9 instance of __EventFilter as $FSC
10 {
11     Name = "ScenarioCreation";
12     Query = "SELECT * FROM __InstanceCreationEvent "
13             "WHERE TargetInstance ISA 'MSFT_UCScenario' "
14             "AND TargetInstance.Id = 'ExampleCorrelationScenario'";
15     QueryLanguage = "WQL";
16 };
17
18 instance of __EventFilter as $FSM
19 {
20     Name = "ScenarioModification";
21     Query = "SELECT * FROM __InstanceModificationEvent "
22             "WHERE TargetInstance ISA 'MSFT_UCScenario' "
23             "AND TargetInstance.Id = 'ExampleCorrelationScenario'";
24     QueryLanguage = "WQL";
25 };
26
27 instance of __EventFilter as $FBOOT
28 {
29     Name = "OnBoot";
30     Query = "SELECT * FROM MSFT_TransientRebootEvent ";
31     QueryLanguage = "WQL";
32 };
33
34 instance of __EventFilter as $FA
35 {
36     Name = "EventAFilter";
37     Query = "SELECT * FROM __InstanceModificationEvent "
38             "WHERE TargetInstance ISA 'ExampleClass' "
39             "AND TargetInstance.Name = 'A'";
40     QueryLanguage = "WQL";

```

};

instance of \_\_EventFilter as \$FB

{

    Name = "EventBFilter";

    Query = "SELECT \* FROM \_\_InstanceModificationEvent "

        "WHERE TargetInstance ISA 'ExampleClass' "

        "AND TargetInstance.Name = 'B'";

    QueryLanguage = "WQL";

};

instance of \_\_EventFilter as \$FTE

{

    Name = "TimerExpiredEvent";

    Query = "SELECT \* FROM MSFT\_TransientEggTimerEvent "

        "WHERE Object ISA \"ExampleCorrelationState\" "

        "AND Object.Scenario = 'ExampleCorrelationScenario'";

    QueryLanguage = "WQL";

};

Defining the bindings:

instance of \_\_FilterToConsumerBinding

{

    Filter = \$FSC;

    Consumer = \$UI;

};

instance of \_\_FilterToConsumerBinding

{

    Filter = \$FSM;

    Consumer = \$UI;

};

instance of \_\_FilterToConsumerBinding

{

    Filter = \$FBOOT;

    Consumer = \$UI;

};



```

1      instance of __FilterToConsumerBinding
2      {
3          Filter = $FA;
4          Consumer = $UA;
5      };

```

```

6      instance of __FilterToConsumerBinding
7      {
8          Filter = $FB;
9          Consumer = $UB;
10     };

```

```

11     instance of __FilterToConsumerBinding
12     {
13         Filter = $FTE;
14         Consumer = $UTE;
15     };

```

When creating this updating consumer scenario, the activation can be triggered to occur using the MSFT\_UCScenario instance. Since, in this example, the system would have \$CI consumer tied to the creation of the Scenario instance, the following instantiation would cause the initialization to occur:

```

16     instance of MSFT_UCScenario
17     {
18         Id = "ExampleCorrelationScenario";
19         Name = "ExampleCorrelationScenario";
20     };

```

This instance helps the updating consumer provider determine how state instances relate to the scenario:

```

21     instance of MSFT_UCScenarioAssociationInfo
22     {
23         Id = "StateAssociation";
24         Scenario = "ExampleCorrelationScenario";
25         Query = "SELECT * FROM ExampleCorrelationState "
                "WHERE Scenario = 'ExampleCorrelationScenario'";
26     };

```

Finally, an example filter to determine when both events occurred within the windows:

```
instance of __EventFilter as $FBOTH
{
    Name = "BothEventsOccurred";
    Query = "SELECT * FROM __InstanceModificationEvent "
           "WHERE TargetInstance ISA \"ExampleCorrelationState\" "
           "AND TargetInstance.ReceivedEventA = TRUE "
           "AND TargetInstance.ReceivedEventB = TRUE "
           "AND TargetInstance.Scenario = 'ExampleCorrelationScenario' ";
    QueryLanguage = "WQL";
};
```

To create a custom event that is triggered when this condition is met, then the user can subscribe an updating consumer to it:

```
instance of MSFT_UpdatingConsumer as $SUBOTH
{
    Id = "BothEventsOccurred";
    Scenario = "ExampleCorrelationScenario";
    Commands = { "INSERT INTO BothAandBEvent ( Name ) "
                "( 'ExampleCorrelationScenario' ) " };
};

instance of __FilterToConsumerBinding
{
    Filter = $FBOTH;
    Consumer = $SUBOTH;
};
```

Fig. 8 illustrates an example of a suitable operating environment in which the event correlation system and method may be implemented. The illustrated operating environment is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Other well-known computing systems, environments, and/or

1 configurations that may be suitable for use with the invention include, but are not  
2 limited to, personal computers, server computers, hand-held or laptop devices,  
3 multiprocessor systems, microprocessor-based systems, programmable consumer  
4 electronics, gaming consoles, cellular telephones, network PCs, minicomputers,  
5 mainframe computers, distributed computing environments that include any of the  
6 above systems or devices, and the like.

7 Fig. 8 shows a general example of a computer 800 that can be used in  
8 accordance with the invention. Computer 800 is shown as an example of a  
9 computer that can perform the various functions described herein. Computer 800  
10 includes one or more processors or processing units 802, a system memory 804,  
11 and a bus 806 that couples various system components including the system  
12 memory 804 to processors 802.

13 The bus 806 represents one or more of any of several types of bus  
14 structures, including a memory bus or memory controller, a peripheral bus, an  
15 accelerated graphics port, and a processor or local bus using any of a variety of  
16 bus architectures. The system memory 804 includes read only memory (ROM)  
17 808 and random access memory (RAM) 810. A basic input/output system (BIOS)  
18 812, containing the basic routines that help to transfer information between  
19 elements within computer 800, such as during start-up, is stored in ROM 808.  
20 Computer 800 further includes a hard disk drive 814 for reading from and writing  
21 to a hard disk, not shown, connected to bus 806 via a hard disk drive interface 815  
22 (e.g., a SCSI, ATA, or other type of interface); a magnetic disk drive 816 for  
23 reading from and writing to a removable magnetic disk 818, connected to bus 806  
24 via a magnetic disk drive interface 819; and an optical disk drive 820 for reading  
25 from and/or writing to a removable optical disk 822 such as a CD ROM, DVD, or

1 other optical media, connected to bus 806 via an optical drive interface 823. The  
2 drives and their associated computer-readable media provide nonvolatile storage  
3 of computer readable instructions, data structures, program modules and other data  
4 for computer 800. Although the exemplary environment described herein employs  
5 a hard disk, a removable magnetic disk 818 and a removable optical disk 822, it  
6 will be appreciated by those skilled in the art that other types of computer readable  
7 media which can store data that is accessible by a computer, such as magnetic  
8 cassettes, flash memory cards, random access memories (RAMs), read only  
9 memories (ROM), and the like, may also be used in the exemplary operating  
10 environment.

11 A number of program modules may be stored on the hard disk, magnetic  
12 disk 818, optical disk 822, ROM 808, or RAM 810, including an operating system  
13 828, one or more application programs 830, other program modules 832, and  
14 program data 834. A user may enter commands and information into computer  
15 800 through input devices such as keyboard 836 and pointing device 838. Other  
16 input devices (not shown) may include a microphone, joystick, game pad, satellite  
17 dish, scanner, or the like. These and other input devices are connected to the  
18 processing unit 802 through an interface 826 that is coupled to the system bus  
19 (e.g., a serial port interface, a parallel port interface, a universal serial bus (USB)  
20 interface, etc.). A monitor 842 or other type of display device is also connected to  
21 the system bus 806 via an interface, such as a video adapter 844. In addition to the  
22 monitor, personal computers typically include other peripheral output devices (not  
23 shown) such as speakers and printers.

24 Computer 800 operates in a networked environment using logical  
25 connections to one or more remote computers, such as a remote computer 846.

1 The remote computer 846 may be another personal computer, a server, a router, a  
2 network PC, a peer device or other common network node, and typically includes  
3 many or all of the elements described above relative to computer 800, although  
4 only a memory storage device 848 has been illustrated in Fig. 8. The logical  
5 connections depicted in Fig. 8 include a local area network (LAN) 850 and a wide  
6 area network (WAN) 852. Such networking environments are commonplace in  
7 offices, enterprise-wide computer networks, intranets, and the Internet. In certain  
8 embodiments, computer 800 executes an Internet Web browser program (which  
9 may optionally be integrated into the operating system 828) such as the "Internet  
10 Explorer" Web browser manufactured and distributed by Microsoft Corporation of  
11 Redmond, Washington.

12 When used in a LAN networking environment, computer 800 is connected  
13 to the local network 850 through a network interface or adapter 854. When used  
14 in a WAN networking environment, computer 800 typically includes a modem 856  
15 or other means for establishing communications over the wide area network 852,  
16 such as the Internet. The modem 856, which may be internal or external, is  
17 connected to the system bus 806 via a serial port interface 826. In a networked  
18 environment, program modules depicted relative to the personal computer 800, or  
19 portions thereof, may be stored in the remote memory storage device. It will be  
20 appreciated that the network connections shown are exemplary and other means of  
21 establishing a communications link between the computers may be used.

22 Computer 800 typically includes at least some form of computer readable  
23 media. Computer readable media can be any available media that can be accessed  
24 by computer 800. By way of example, and not limitation, computer readable  
25 media may comprise computer storage media and communication media.

Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other media which can be used to store the desired information and which can be accessed by computer 800. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

The invention has been described in part in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Typically the functionality of the program modules may be combined or distributed as desired in various embodiments.

For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the data processor(s) of the computer.

Although the description above uses language that is specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the invention.